Cost Effective Development of Usable Systems; Gaps between HCI and SE

Eelke Folmer, Jan Bosch
Department of Mathematics and Computing Science
University of Groningen, PO Box 800, 9700 AV, The Netherlands
mail@eelke.com, Jan.Bosch@cs.rug.nl

Abstract

Usability is considered an important quality attribute for software systems. To ensure a particular level of usability, a certain amount of time and money have to be invested; however this has proven to be expensive. Most of the costs spent on usability are spent after an initial development e.g. during maintenance. These high costs often prevent developers from meeting all the usability requirements. The challenge is therefore to cost effectively develop usable software e.g. minimize the costs & time spent on usability. We believe architecture analysis of usability is an important tool to achieve this. Our experiences with software architecture analysis of usability allowed us to identify a series of problems that explain why usability is not achieved cost effectively in current software development practice.

1. Introduction

Cost, quality and time-to-market are three main concerns that make software engineering projects true challenges. Costs should be minimized to increase profit and market share, quality should be maximized to attract and satisfy customers and time-to-market should be minimal to reach the market before competitors do[3].

Decisions which affect these concerns have a tradeoff cost, if more features are added to the product, quality must drop or time to market must slip. If time to market is cut, features must drop or quality must drop [6]. Cost, quality and time to market are related; improvements in one may affect at least one of the others negatively.

Usability is considered as an essential part of software quality [8]. To develop a usable system one must be willing to invest a certain amount of time and money. In relative new markets, such as web based software, minimizing time to market and costs have often been preferred at the expense of usability. This poses to become a problem in the future because, as users become more critical, poor usability becomes a major barrier to the success of new commercial software applications.

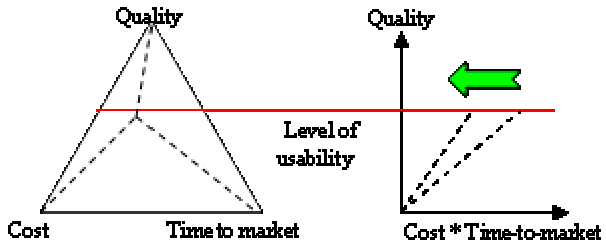

Figure 1: tradeoffs during design

Most of the costs spent on usability are spent *after* initial development e.g. during maintenance. Studies of software engineering projects reveal that organizations spend a relatively large amount of time and money on fixing usability problems. Several studies have shown that 80% of total maintenance costs are related to problems of the user with the system [10]. Among these costs, 64% are related to usability problems [11].

The challenge is therefore to *cost effectively* develop usable software. Minimizing the costs & time that are spent on usability improves usability of system because these high costs often prevent developers from meeting all the usability requirements. Cost effective development makes it cheaper to assure a particular level of usability or get a more usable system for the same investment (see Figure 1).

Based upon successful experiences with architectural assessment of maintainability as a tool for cost effective developing maintainable software, we developed and promoted the use of architectural assessment of usability [12,13] as an important tool to cost effectively development usable software. Our experiences with software architecture analysis of usability in several case studies have led us to identify a series of problems that in explain why usability is not achieved cost effectively in current software development practice. Some of these problems can be considered as gaps between HCI & SE

though not all problems we present here are necessarily gaps between both communities, but rather a failure and shortcoming of the current practice of one community. The next sections [2-9] discuss the problems we identified. This paper is concluded in section 10.

2. Usability requirements specification

In our experience usability requirements are often poorly specified. In all cases we performed, apart from some general usability guidelines [6] that had been stated in the functional requirements, no clearly defined and verifiable usability requirements had been collected nor specified. Most software developing companies still underestimate the importance of usability and usability engineering and postpone the activity of usability requirements collection till there is a running system. Usability is often not defined as an explicit project goal.

Even if usability requirements are specified, they are specified on a rather abstract level. A usability requirement such as: 'the system should be easy to learn' does not state anything about users, contexts of use or tasks for which this requirement should hold. Existing usability techniques such as such as interviews, group discussions or observations [1,4,14,15] typically already provide information such as representative tasks, users and contexts of use

The reason for this abstract specification is that traditionally usability requirements have been specified such that these can be verified for an implemented system. However, such requirements are largely useless in a forward engineering process. For example, we could say that a goal for a system is that it should be easy to learn, or that new users should require no more than 30 minutes instruction, however, a requirement at this level does not help guide the design process. Such requirements can only be measured when the system has been completed. Usability requirements need to take a more concrete form expressed in terms of the solution domain to influence design.

3. Limitation of requirements engineering techniques

Software engineers in general have few techniques available for predicting the quality attributes of a software system before the system itself is available. This is especially hard for usability since in order to do a usability evaluation, most existing techniques require at least an interactive prototype and a representative set of users present to assess the usability of a system [12]. Next to that usability-engineering techniques have only a limited ability to capture or predict all usability requirements. Users themselves lack understanding of their own requirements. No sooner do they work with a first version of the software do they realize how the system is going to be used. Usability experts miss about half of the problems that real users experience using traditional techniques [16].

Some techniques such as rapid prototyping [1] allow for early testing, for example by using a prototype or simulation of an interface. Early prototyping, even on paper, of what the customer's experience will be like, always is valuable. However, prototypes have a limited ability to model the application architecture, since they only model the interface. Interaction issues such as the time it takes to perform a specific task or system properties such as reliability have a great influence on the level of usability. Such issues may be hard to simulate with a prototype. Some usability requirements will therefore not be discovered until the software has been deployed.

4. Usability requirements change during development

During or after the development usability requirements change. The context in which the user and the software operate is continually changing and evolving, sometimes users may find new uses for a product, for which the product was not originally intended. New features get added to an existing software product during product evolutions which have different usability requirements. In one of our case studies (a content management system), after the product had been developed, new features (support & manipulation for streaming video) were considered to be developed that would be built in the existing application framework. The software architecture did not sufficiently support these new usability requirements. It was decided because of that (and other reasons) not to add these new features to the existing application but rather develop these features as a standalone application. This example shows that it is hard or even impossible to capture all possible (future) usability requirements during initial design [8].

5. Lack of assessment/ design techniques

In general software developers must develop their software in such a way that the software is usable by all the relevant stakeholders.

1. In order to do so they need to be able to *extract* requirements with respect to usability from users.
2. They need to have techniques to *realize* these requirements.

3. They need to be able to *assess* whether the resulting product actually meets the usability requirements.

In the ideal situation these steps should be followed at every step of the development process (as far as applicable). *Extract requirements*: During initial design, but also during the later stages to verify whether the usability requirements acquired during requirements analysis are still valid. *Realize requirements*: during all stages of design. *Assessment*: Not only when we have a running system, but during all stages of development even during requirements analysis. For example, verifying the collected set of usability requirements versus usability heuristics or interface guidelines.

Unfortunately the ideal situation is often far from current practice. Figure 2 shows which (requirements collection/ assessment/ design) techniques are applied at which stages. If a particular process cannot be applied at a particular stage (such as realization at requirements analysis) the arrows are left out. The color of the arrows indicates the *usage* of these techniques in current practice

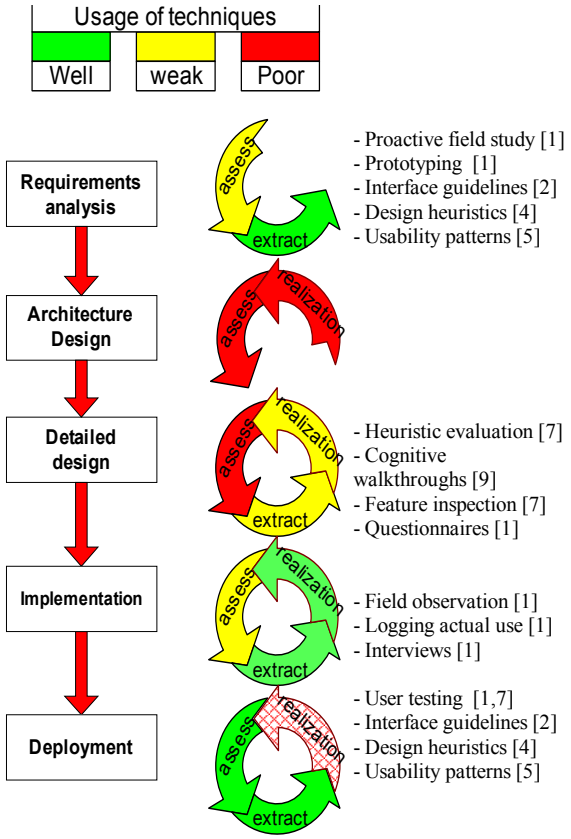

Figure 2: Techniques applied at each stage of the development process

This figure shows that:

- *Extraction* is often only done during requirements analysis or during implementation/deployment (when there is a working system and a representative set of users available).
- *Realization* of usability requirements is often only done during detailed design (interaction design) and implementation. After deployment it is sometimes too expensive to fix fundamental usability issues.
- *Assessment* is often only done during requirements analysis (verification of usability requirements versus interface / usability guidelines) or when we have running system prototype (e.g. during detailed design & implementation & deployment)

Figure 2 shows three problems with current design.

- No assessment is done during the early stages of design.
- No realization is done during the early stages of design
- Realization of usability is often too expensive to be done during the later stages of development.

The first problem is caused because software engineers in general have few techniques available for predicting or realizing the quality attributes of a software system before the system itself is available. Most engineering disciplines provide techniques and methods that allow one to assess and test quality attributes of the system under design. For example for maintainability assessment code metrics [17] have been developed. In [12] an overview is provided of usability evaluation techniques that can be used during software development Some of the more popular techniques such as user testing [7], heuristic evaluation [1] and cognitive walkthroughs [9] can be used during several stages of development, however there are no assessment techniques that focus on assessment of usability during the early stages of design (e.g. software architecture design).

The second problem is caused by that usability requirements are often specified in a format that does not guide architectural design. During architectural design decisions are made that are the hardest to revoke. The third problem is related to the second problem and is a serious problem that is responsible for the high costs of usability development. The cause for this problem will be discussed in the next sections.

6. The impact of software architecture on usability

Discovering requirements late is a problem inherent to all software development and is not something that can easily be avoided. The real problem is that it often proves to be hard and expensive to make the necessary changes to a running system to improve its usability.

The software engineering community often considers usability to be primarily a property of the presentation of information; the user interface [4]. In web applications

the user interface is almost always described as the top layer. This implies (intentional or not) that this layer is simpler than the underlying layers and also the least consequential to the overall architecture. If usability needs to be improved, changes to the interface can easily be applied after user testing, which does not affect the rest of the application. It's easy for engineers to believe, because it facilitates the notion that somehow, as long as everything is done well at the lower layers, the interface layer will be easy to manage [6].

Engineers assume that a separation of the interface from the rest of the application can still ensure usability. We believe this is a false assumption. Usability is determined by many factors not only the interface but also issues such as:

- Information architecture: how is the information presented to the user?
- Interaction architecture: how is functionality presented to the user?
- System quality attributes: usability is considered as a part of software quality but also results from other quality attributes such as performance and reliability.

Architecture design does affect all these issues. For example the quality attributes such as performance or reliability are to a considerable extent defined by the software architecture. The software architecture also has major impact on the interaction & information architecture. Designing a usable system is more than ensuring a usable interface; a slow and buggy system architecture with a usable interface is not considered usable on the other hand the most reliable and performing system architecture is not usable if the user can't figure out how to use the system. Software engineers do not realize usability should also be realized at the architectural level.

7. The impact of usability on software architecture.

As mentioned before the software engineering community often considers usability to be primarily a property of the presentation of information; the user interface [4]. A result of this assumption is that interface design is often postponed to the later stages of development. There are two risks with this approach:

- Assumptions may be built into the design of the architecture that may unknowingly affect the interface design. In one of our case studies (a large content management system) we identified that the layout of a page (users had to fill in a form) was determined by the XML definition of a specific object. When users had to insert data, the order in which particular fields had to be filled in turned out to be very confusing. This is just one of many examples where we identified that the architecture placed constraints on interface design. The interface should not be designed as last but should be developed as early as possible preferably even during requirements analysis (interface prototypes) to identify such issues.
- Assumptions may be built in the interface that are not supported by the architecture:
    - Interaction issues (such as the support for a wizard or undo).
    - Information architecture issues (such as a separation of data from presentation).
    - Interface issues (such as visual consistency).

These are examples of usability solutions that increase the usability of systems but are extremely hard to retrofit in the architecture. Or to put it in other words had we taken these issues into account during architectural design these usability solutions could much easier be supported than trying to build them in when the systems has been finished. Our research [18] argues that to effectively implement such solutions they require the architecture to be restructured. The cost of restructuring the system during the later stages of development has proven to be several orders of magnitude higher than the costs of an initial development, this making the total costs spent on usability very high.

8. Technological view drives design.

As pointed out by [6] most (architectural) design is very "technology" driven. E.g. a software product is often seen as a set of *features* rather then a set of *user experiences*. In the case studies we performed we identified that software architects had already selected technologies (e.g. *features*) and had already developed a first version of the system before they decided to include the user in the loop. After that it was already too late to make fundamental changes required by usability.

The best software comes from teams, or from team leaders, that are able to see the work from multiple perspectives, balancing them in accordance with the project goals, and the state of the project at any given time. When development is dominated by a technological perspective, it's natural for software engineers to make decisions that optimize technological considerations over all others. The technological view of a product is only one of many views, It takes the right combination of perspectives to achieve great products [6]

Since software engineers are not usability experts and usability experts are not software engineers, the responsibilities of defining and collecting usability requirements should be separated from the architectural

design responsibilities. This very much depends on the size of the software developing organization but in the case studies we performed (varying from small to medium sized organizations) only at one case study these responsibilities were divided. A better balancing of the different views of the system (and hence better usability) is achieved when software is designed in multi disciplinary teams.

9. Software architecture analysis is an ad hoc activity

Because quality attributes are to a considerable extent defined by the software architecture, the design and use of an explicitly defined software architecture has received increasing amounts of attention during the last decade. Generally, three arguments for defining an architecture are used [19]. First, it provides an artifact that allows discussion by the stakeholders very early in the design process. Second, it allows for early assessment of quality attributes [20,21]. Finally, the design decisions captured in the software architecture can be transferred to other systems.

Software architecture analysis is an important tool to get feedback during the early stages of design. A software architecture description such as a decomposition of the system into components and relations with its environment may provide information on the support for particular quality attributes. Specific relationships between software architecture (such as -styles, -patterns etc) and quality attributes (maintainability, efficiency) have been described by several authors. [22,23,21]. For example [22] describes the architectural pattern layers and the positive effect this pattern may have on exchangeability and the negative effect it may have on efficiency. For usability these relationships with software architecture need to be investigated and described so they can be used to inform architectural design.

As identified by [24] architectural assessment is least applied in practice. Architecture analysis is mostly performed on an ad-hoc basis. The assessment is not solidly embedded in the development process and there is no or little integration & cooperation with existing usability requirements collection techniques.

Software engineers have few techniques available for predicting the quality attributes of a software system before the system itself is available. An increased awareness of the importance of the architectural impact of usability could lead to the development of tools & assessment technique that assist the software architect in designing an architecture that supports usability. If such a technique is an integral part of the development process, earlier phases or activities would result in the necessary information required for the analysis. For example, specified usability requirements and architectural descriptions.

10. Conclusions

Ensuring a particular level of software quality (e.g. usability) proves to be very expensive. The high costs often prevent developers from meeting all the usability requirements leading to systems that are not usable. Cost effective usability development is an important tool to improve the usability of systems.

Architecture analysis of usability is an important tool to cost effectively develop usable software. In the context of experiences with software architecture analysis of usability we have identified several problems that explain why usability is not achieved cost effectively:

Some usability requirements will not be discovered until the software has been implemented/deployed. This is caused by the following:

- Usability requirements are often weakly specified.
- Usability requirements engineering techniques have only limited ability to capture all requirements.
- Usability requirements change during development.
- Usability testing is only done at the end because there are no early assessment tools.

Discovering requirements late is a problem inherent to all software development and is not something that can be easily avoided. The real problem is that it often proves to be hard and expensive to make the necessary changes to developed system to improve its usability. Reasons for why this is so hard:

- Usability does also depend on issues such as the information architecture, the interaction architecture and other quality attributes that are all determined by the software architecture. Usability should therefore also be realized at the architectural level.
- Many of the necessary usability changes to the system cannot be easily be accommodated by the software architecture.

Software architects are not aware of the relationship between usability and software architecture because:

- Design is technology driven.

The costs of restructuring the system during the later stages of development has proven to be several orders of magnitude higher than the costs of an initial development [21]. This explains why organizations are spending so much time and money on usability during maintenance. In essence usability is a maintainability problem. Designing a well performing, reliable and flexible architecture that can support unforeseen usability requirements is quite a challenge. Since the architecture plays such a major role in the usability of a system, early

assessment could solve some of the problems we discovered however:

- We lack early assessment tools.
- Usability requirements are often weakly specified.
- Software architecture analysis in general is an ad-hoc activity.

The other problems or gaps may be addressed by raising the awareness of the importance of the relationship between usability and software architecture but also by raising the importance of usability as the most import quality attribute and software architecture as an important instrument to fulfill this attribute. By raising the awareness of this relationship eventually software engineers and usability engineers must recognize the need for a closer integration of practices and techniques leading to cost effective development of usable systems.

11. References

[1] J. Nielsen, *Usability Engineering*, Academic Press, Inc, Boston, MA., 1993.

[2] ISO, ISO 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability., 1994.

[3] P. O. Bengtsson, *Architecture-Level Modifiability Analysis*, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Sweden, 2002.

[4] D. Hix and H. R. Hartson, *Developing User Interfaces: Ensuring Usability Through Product and Process.*, John Wiley and Sons, 1993.

[5] M. Welie, *GUI Design patterns*, http://www.welie.com/

[6] S. Berkun, *The list of fourteen reasons ease of use doesn't happen on engineering projects*, http://www.uiweb.com/issues/issue22.htm

[7] J. Nielsen, *Heuristic Evaluation.*, in Usability Inspection Methods., Nielsen, J. and Mack, R. L., John Wiley and Sons, New York, NY., 1994.

[8] ISO, ISO 9126-1 Software engineering - Product quality - Part 1: Quality Model, 2000.

[9] C. Wharton, J. Rieman, C. H. Lewis, and P. G. Polson, *The Cognitive Walkthrough: A practitioner's guide.*, in Usability Inspection Methods, Nielsen, Jacob and Mack, R. L., John Wiley and Sons, New York, NY., 1994.

[10] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, NY, 1992.

[11] T. K. Landauer, *The Trouble with Computers: Usefulness, Usability and Productivity.*, MIT Press., Cambridge, 1995.

[12] E. Folmer and J. Bosch, *Architecting for usability; a survey*, Elsevier, 2002, pp. 61-78.

[13] E. Folmer, J. v. Gurp, and J. Bosch, Scenario-based Assessment of Software Architecture Usability, *icse 2003 workshop Bridging the Gaps Between Software Engineering and Human-Computer Interaction*, 2003.

[14] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA, 1998.

[15] J. T. Hackos and J. C. Redish, *User and Task Analysis for Interface Design*, John Wiley and Sons, Inc. New York, 1998.

[16] D. L. Cuomo and C. D. Bowen, *Understanding usability issues addressed by three user-system interface evaluation techniques*, Elsevier, 1994, pp. 86-108.

[17] W. Li and S. Henry, *OO Metrics that Predict Maintainability*, Elsevier, 1993, pp. 111-122.

[18] E. Folmer, J. v. Gurp, and J. Bosch, *Investigating the Relationship between Usability and Software Architecture* , Wiley, 2003, pp. 0-0.

[19] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison Wesley Longman, Reading MA, 1998.

[20] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, The Architecture Tradeoff Analysis Method, *Proceedings of ICECCS'98*, 8-1-1998.

[21] J. Bosch, *Design and use of Software Architectures: Adopting and evolving a product line approach*, Pearson Education (Addison-Wesley and ACM Press), Harlow, 2-1-2000.

[22] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley and Son Ltd, 6-1-1996.

[23] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns elements of reusable object-orientated software.*, Addison -Wesley, 1995.

[24] N. Lassing, P. O. Bengtsson, H. van Vliet, and J. Bosch, *Experiences with ALMA: Architecture-Level Modifiability Analysis*, Elsevier, 2002, pp. 47-57.